Overview

I. Introduction and getting started
II. Growth of functions and asymptotic notations
III. Divide-and-conquer recurrences and the master theorem
IV. Divide-and-conquer algorithms
V. Greedy algorithms
VI. Dynamic programming
VII. Graph algorithms
VII. NP-completeness
I. Introduction and Getting Started
Algorithm is a tool for solving a well-specified computational problem.

A poem by D. Berlinski in “Advent of the Algorithm”

*In the logician’s voice:*

> an algorithm is
> a finite procedure,
> written in a fixed symbolic vocabulary
> governed by precise instructions,
> moving in discrete steps, 1, 2, 3, ...
> whose execution requires no insight, cleverness,
> intuition, intelligence, or perspicuity
> and that sooner or later comes to an end.
Introduction

- Basic questions about an algorithm
  1. Does it halt?
  2. Is it correct?
  3. Is it fast?
  4. How much memory does it use?
  5. How does data communicate?

- Algorithms as a technology
  TED Talk: *How Algorithms Shape Our World* by Kevin Slavin, July 2011
Getting started: example 1

- Problem statement: computing the $n$th Fibonacci number $F_n$
- Definition of Fibonacci numbers:
  
  \[
  F_0 = 0,
  F_1 = 1,
  F_n = F_{n-1} + F_{n-2} \quad \text{for } n \geq 2
  \]

- Algorithms:
  1. Recursion ("top-down")
  2. Iteration (memoization) ("bottom-up")
  3. Divide-and-conquer
  4. Approximation
Getting started: example 2

▶ Problem statement:

Input: a sequence of $n$ numbers $\langle a_1, a_2, \ldots, a_n \rangle$
Output: a permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the
$a$-sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

In short, sorting

▶ Algorithms:

1. Insertion sort
2. Merge sort
Getting started: example 2

Insert sort algorithm

- Idea: incremental approach
- Pseudocode

```plaintext
InsertionSort(A)
1 n = length(A)
2 for j = 2 to n
3   key = A[j]
4 // insert 'key' into sorted array A[1...j-1]
5   i = j-1
6   while i > 0 and A[i] > key do
7     A[i+1] = A[i]
8     i = i-1
9   end while
10   A[i+1] = key
11 end for
12 return A
```
Getting started: example 2

Remarks:

- Correctness: argued by “loop-invariant” (a kind of induction)
- Complexity analysis:
  - best-case
  - worst-case
  - average-case
- Insertion sort is a “sort-in-place”, no extra memory necessary
- Importance of writing a good pseudocode = “expressing algorithm to human”
- There is a recursive version of insertion sort, see Homework 1.
Getting started: example 2

Merge sort algorithm

- Idea: divide-and-conquer approach
- Pseudocode

```plaintext
MergeSort(A,p,r) // Merge-sort of array A[p..r]
1   if p < r then // check for base case
2       q = flooring((p+r)/2) // divide
3       MergeSort(A,p,q) // conquer
4       MergeSort(A,q+1,r) // conquer
5       Merge(A,p,q,r) // combine
6   end if
```
Getting started: example 2

- Pseudocode, cont'd

```plaintext
Merge(A,p,q,r)

n1 = q-p+1; n2 = r-q
for i = 1 to n1 // create arrays L[1...n1+1] and R[1...n2+1]
    L[i] = A[p+i-1]
end for
for j = 1 to n2
    R[j] = A[q+j]
end for
L[n1+1] = infty; R[n2+1] = infty // mark the end of arrays L and R
i = 1; j = 1
for k = p to r // Merge arrays L and R to A
    if L[i] <= R[j] then
        A[k] = L[i]
        i = i+1
    else
        A[k] = R[j]
        j = j+1
    end if
end for
```
Getting started: example 2

- Merge sort is a divide-and-conquer algorithm consisting of three steps: divide, conquer and combine.
- To sort the entire sequence $A[1...n]$, we make the initial call
  $$\text{MergeSort}(A,1,n)$$
  where $n = \text{length}(A)$.
- Complexity analysis:
  $$T(n) = 2T\left(\frac{n}{2}\right) + n - 1 = O(n \log(n))$$